Claims

1. (original) A system for detecting a data race in a concurrent program, the system comprising:

a program sequentializer module configured to accept a concurrent program as input and create as output a sequential program having assertions;

wherein the assertions cause an error message to be produced when the concurrent program contains a data race.

2. (original) The system of claim 1, further comprising a sequential program analyzer module which analyzes the sequential program and produces error messages if assertions are not met and wherein the assertions are created to be checked by the sequential program analyzer.

3. (original) The system of claim 1, wherein the sequential program utilizes a single runtime stack and the program sequentializer module is further configured to add data structures to the received code, the added data structures at least comprising:

a multiset of thread pointers which comprises pointers to threads which have been created but have not yet been scheduled on the runtime stack; and

a global boolean exception variable which, when set, causes the sequential program to remove from the runtime stack the currently-executing thread.

4. (original) The system of claim 3, wherein:

the multiset of thread pointers is limited to a maximum number of pointers; and

the added data structures further comprise a global multiset size variable, which indicates the maximum number of pointers.

5. (original) The system of claim 3, wherein:

the program sequentializer module is further configured to receive an indication of a target variable which will be analyzed for data races; and

the added data structures further comprise a global access variable which indicates, for the analyzed variable, the current type of access being requested of the variable.

6. (original) The system of claim 5, wherein the program sequentializer module is further configured to insert instrumentation into the received code, the instrumentation at least comprising:

a scheduling function which selects a thread pointer from the multiset and schedules the

thread indicated by the pointer on the runtime stack;

an exception macro which sets the global boolean exception variable and causes an executing thread to be removed from the runtime stack;

a read-checking function, which checks to see that the global access variable does not indicate that the target variable is being written to and then sets the global access variable to indicate that the target variable is being read from; and

a write-checking function, which checks to see that the global access variable does not indicate that the target variable is being read from or written to and then sets the global access variable to indicate that the target variable is being written to.

7. (original) The system of claim 6, wherein the read-checking and write-checking functions contain assertions about the global access variable which can be checked by the sequential program analyzer module.

8. (original) The system of claim 6, wherein the instrumentation is inserted so that it will execute nondeterministically in the sequential program.

9. (original) The system of claim 1, wherein the sequential program output by the sequentializer module is in the form of source code.

10. (original) The system of claim 1, wherein the sequential program output by the sequentializer module is in the form of an abstract syntax tree or a control-flow graph.

11. (original) The system of claim 1, wherein the concurrent program received by the sequential analyzer is in the form of source code.

12. (original) The system of claim 1, wherein the concurrent program received by the sequential analyzer is in the form of an abstract syntax tree or control-flow graph.

13. (original) A method of analyzing a concurrent program for data races, the method comprising:

receiving a concurrent program;

receiving at least one target variable to be analyzed for data races; and

creating a sequential program from the concurrent program, the sequential program containing assertions such that if the assertions are not met, the presence of a data race in the concurrent program for the target variable is indicated.

14. (original) The method of claim 13, wherein:

creating a sequential program comprises adding instrumentation and variables to the

concurrent program which cause the functions of the concurrent program to be executed sequentially; and

the added variables include:

a multiset of thread pointers which comprises pointers to threads which have been started but have not yet been scheduled on the runtime stack of the sequential program; and

a global boolean exception variable which, when set, causes the sequential program to remove from the runtime stack the currently scheduled thread; and

a global access variable which denotes, for the analyzed variable, the current type of access being requested of the variable.

15. (original) The method of claim 14, wherein the added instrumentation includes:

a scheduling function which selects a thread pointer from the multiset and schedules it on the runtime stack;

an exception macro which sets the global boolean exception variable and causes an executing thread to be removed from the runtime stack;

a read-checking function, which checks to see that the global access variable does not indicate that the target variable is being written to and then sets the global access variable to indicate that the target variable is being read from; and

a write-checking function, which checks to see that the global access variable does not indicate that the target variable is being read from or written to and then sets the global access variable to indicate that the target variable is being written to.

16. (original) The method of claim 15, wherein the read-checking and write-checking functions contain assertions about the global access variable which can be checked by a sequential program analyzer.

17. (original) The method of claim 15, wherein the instrumentation is inserted so that it will execute nondeterministically in the sequential program.

18. (currently amended) A computer-readable medium containing instructions which, when executed, check a concurrent program for one or more violations of assertions, the instructions configured to:

receive a concurrent program; and

create a sequential program from the concurrent program, the sequential program containing assertions such that if the assertions are not met, the presence of [[a]] an error in the

concurrent program is indicated.

19. (original) The computer-readable medium of claim 18, wherein:

the instructions configured to create a sequential program comprise instructions configured to add instrumentation and variables which cause the functions of the concurrent program to be executed sequentially; and

the added variables include:

a multiset of thread pointers which comprises pointers to threads which have been started but have not yet been scheduled on the runtime stack of the sequential program; and

a global boolean exception variable which, when set, causes the sequential program to remove from the runtime stack the currently scheduled thread.

20. (original) The computer-readable medium of claim 19, wherein the added instrumentation includes:

a scheduling function which selects a function pointer from the multiset and schedules it on the runtime stack; and

an exception macro which sets the global boolean exception variable and causes an executing thread to be removed from the runtime stack.